Octree Optimization

Al Globus

Report RNR-90-011, July 1990

This work was supported by NASA Contract NAS 2-12961 to Computer Scineces Corporation for the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center.

Copies of this report are available from:
NAS Applied Research Office
Mail Stop T045-1
NASA Ames Research Center
Moffett Field, CA 94035
(415) 604-4332

Octree Optimization

Al Globus
Computer Science Corporation
NASA Ames Research Center
18 July 1990

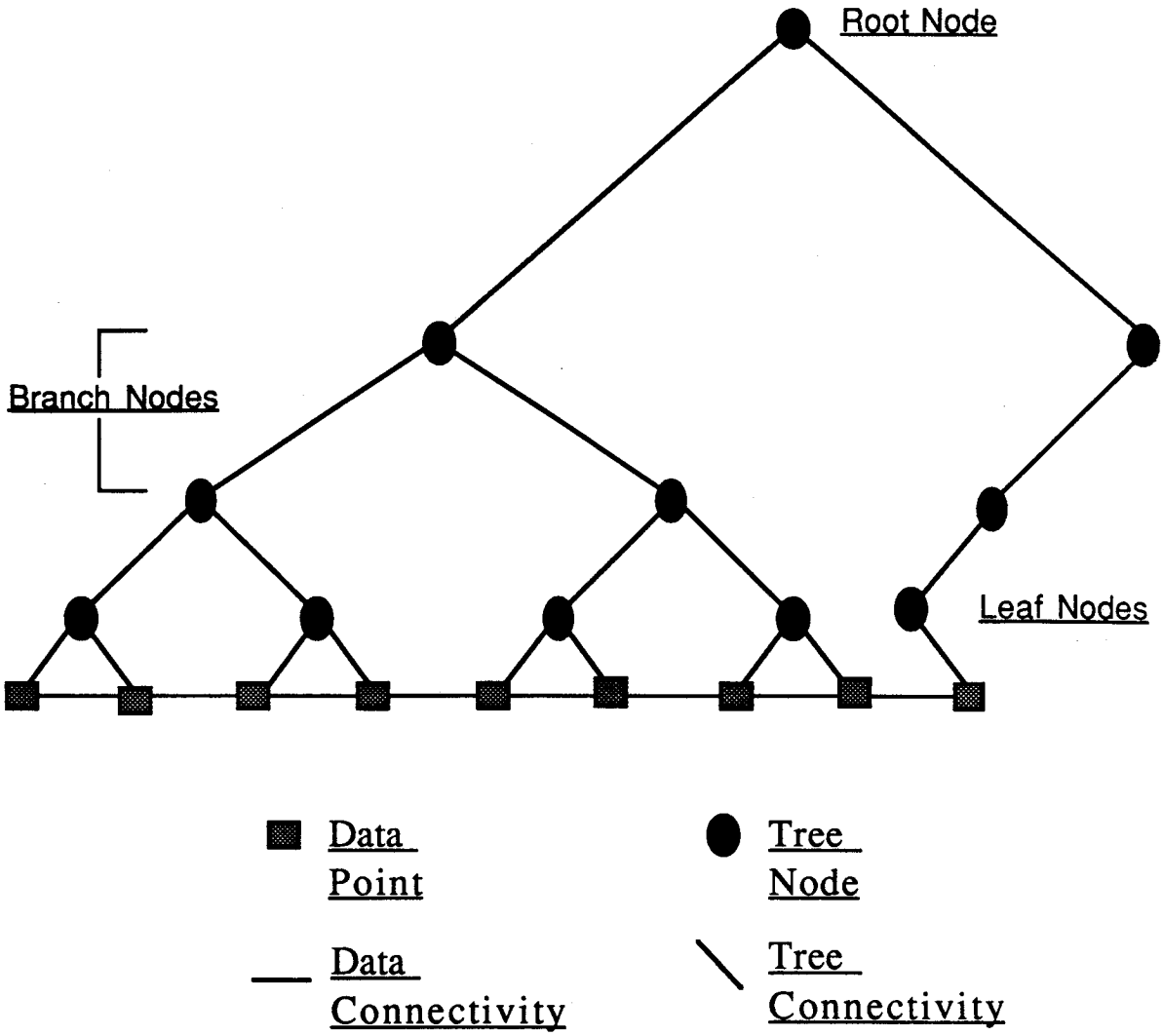
Abstract

A number of algorithms search large, typically 3D, computational spaces for features of interest. Marching cubes isosurface generation described by Lorensen and Cline[1] is an example. The speed of these algorithms is dependent on the time necessary to 1. find the features of interest in the data and 2. to compute over them. This paper describes an optimizing search using octrees to divide computation space. Information stored in the branch nodes is used to prune portions of computation space thus avoiding unnecessary memory references and tests for features of interest. This technique is implemented for marching cubes isosurface generation on computational fluid dynamics data. Numerical experiments were run indicating a factor of 3.8 - 9.0 overall performance increase, measured by stopwatch; and a factor of 3.9 - 9.9 speedup in calculation times as measured by the UNIX times()[2] utility. The overhead is a one time cost of 0.2 - 2.8 the time to compute an average isosurface and O(n) space with a small constant factor.

Introduction

The algorithms whose optimization is addressed here are those that must repeatedly search a large computational space, e.g., a large 3D array of real numbers representing a scalar field, to find features of interest. Algorithms presently using exhaustive search can sometimes be sped up significantly if not all of computational space need be examined in detail.

Octree optimization breaks up computation spaces with an octree to divide the search space. The octree technique, originally developed for solid modeling, has been described by Jackins and Tanimoto[3] and Meagher[4]. In octree optimization, the root node represents all of computational space. It is recursively subdivided into eight children each representing approximately one-eighth of the space. Recursion ceases when each node



Previous work includes a paper by Woodwark[5] who used a similar technique in a different context. The kD tree search techniques developed by Bently[6] in the data base world are closely related. Wilhelm at the University of California at Santa Cruz has also implemented an octree-based computation space optimization technique and plans to apply it to marching cube isosurface generation [personal communication]. Glassner[7] uses octrees in physical space to reduce the number of objects to test for ray trace intersections. Kerlick[8] describes a competitive technique which uses sorting to limit searches for isosurfaces generated using marching

ISOLEV creates function mapped cutting planes by generating level fields, applying the marching cubes algorithm to find isosurfaces (which are always planes) and shading the result using the scalar field of interest. Sweeps are particularly useful for cutting plane displays since, if they are fast enough, the user can get a understanding of the whole 3D space.

ISOLEV was modified to optionally use octree optimization. The modifications were implemented such that precisely the same code is invoked to perform the marching cubes algorithm whether octree optimization is used or not. This is necessary for the validity of the numerical experiments.

Three operations are required for octree optimization implementation:

- Build the tree, necessary when the size of the computational space changes. Each node is made to represent a space as 'cubic' as possible, i.e., the dimensions are as nearly equal to each other as possible.
- Set the min/max values in each node, necessary when the scalar field of interest changes or a new direction for a cutting plane is chosen. The min/max is used to test isovalues to see if the surface passes through the data represented by a node.
- Walk the tree, necessary when the value of the isosurface or location of the cutting plane changes. When a leaf node is reached and the surface is present, run the marching cubes algorithm on the sub-space represented.

The tree is built top down. The root node represents all of computation space. This is divided into eight (approximately) equal sub-spaces - usually by dividing each dimension by two - to produce the root's children. Oddly shaped subspaces are handled by an algorithm described below. This procedure is applied recursively to generate the whole tree. Recursion stops when a node represents less than 32 grid points.

To set the min/max values, the tree is walked and the min/maxes passed up from node to node.

To walk the tree, a function is called with the following parameters:

- Tree node to walk .
- Isovalue.

```
    pv = 1,1,8
else if Dl >= 2Dm >= 2Ds
    pv = 1,2,4
else
    pv = 2,2,2
```

To get the eight subspaces needed for the children, divide each dimension by the appropriate part of pv. If it divides evenly, or the divisor is two, everything is fine. If not, then make all but the last section equal to the dimension size / pv[n]. Then let x = (dimension size mod pv[n]) - 1. Make x sections one larger starting with the second to last and working backwards. This will cause all the sections to be within one of the same size.

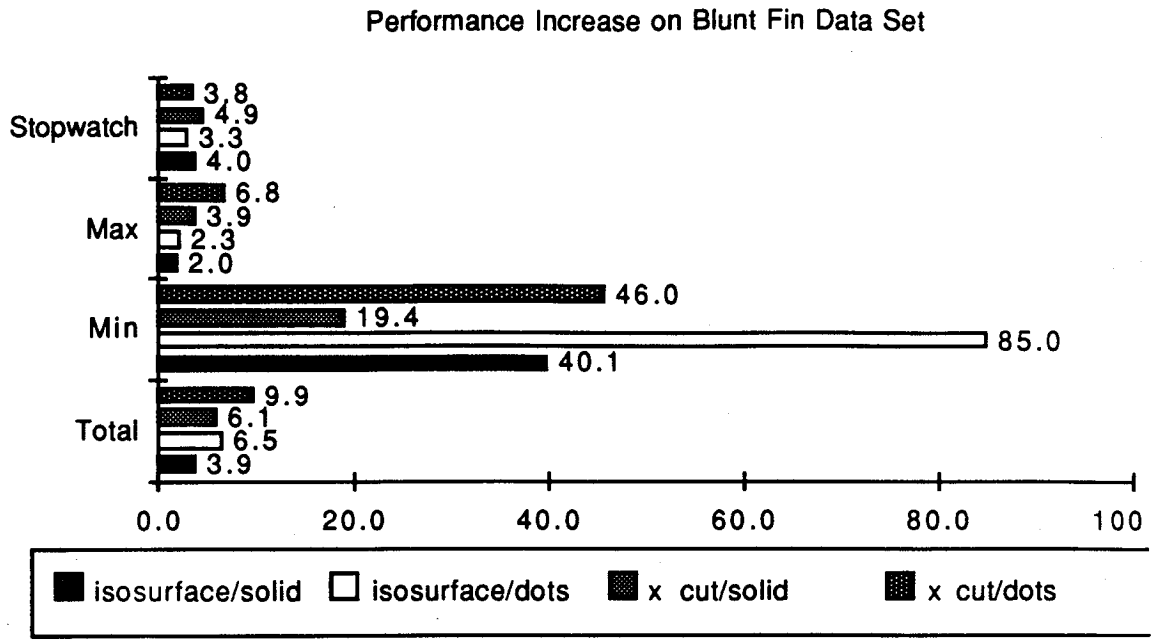
Experiments

A set of computational experiments were run to determine if octree optimization of ISOLEV actually improves performance substantially. Data sets were taken from work performed at the National Aeronautic Simulation facility at NASA Ames Research Center.

Time is the dependent variable, the presence or absence of octree optimization the independent variable. Measures were taken by sweeping a cutting plane through each data set and sweeping isovalues to get isosurfaces through each data set. Each sweep was done in 20-21 steps. In each case, experimental runs with and without optimization were made. Stopwatch time was collected since it is the most important to the user. For stopwatch timed runs, each condition was repeated three times and the median or mean taken since uncontrolled system events can theoretically cause variations in the results. Actual results were very closely clustered.

In a separate experimental run, the UNIX times() utility was used to generate the:

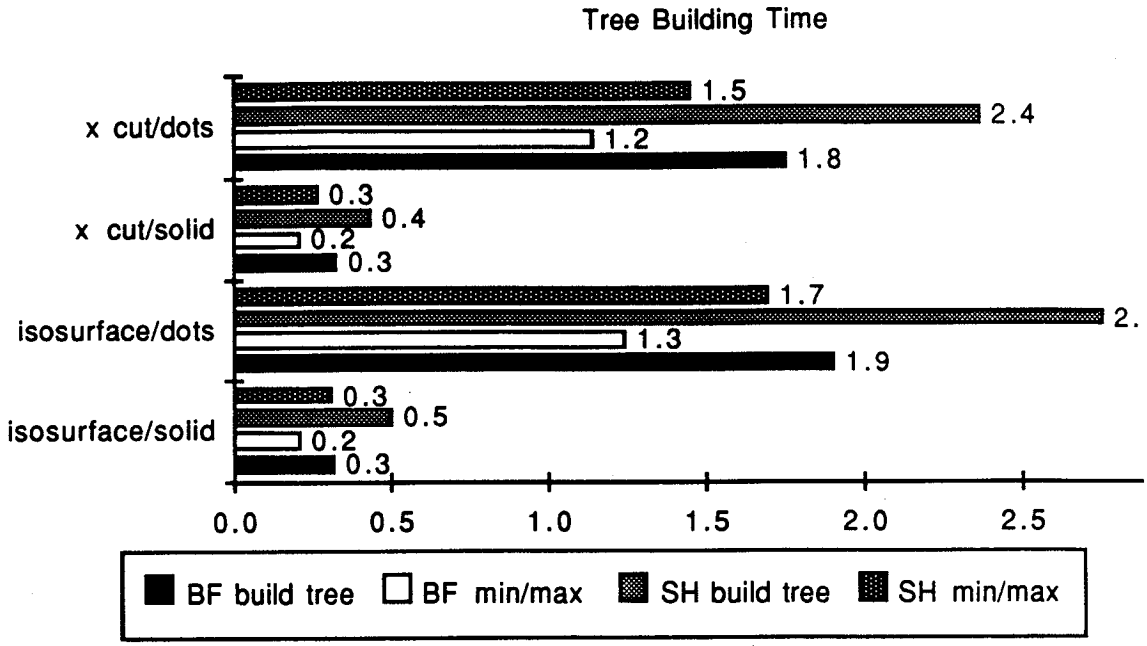
- Total time to compute the surfaces in optimized and unoptimized modes.
- Minimum time to compute one of the surfaces in optimized and unoptimized modes.
- Maximum time to compute one of the surfaces in optimized and unoptimized modes.



Note that improvements are very large for minimum times. The fact that most the search space can be pruned away by the optimization causes this. The opposite effect causes maximum times to be more nearly equal.

Stopwatch timed improvements are not as good since they include time to rende the surfaces and overhead introduced by FAST.

This next chart expresses tree building time as a factor of the average time to generate one unoptimized isosurface (or cutting plane). E.g., building the tree (including generating min/maxes) for the Blunt Fin took 0.3 times as long as unoptimized generation of an average isosurface.



Note that the time penalty for building optimization octrees is quite small. It is paid back within a few isosurfaces.

BF means Blunt Fin. SH means shuttle.

The raw data is in appendix A.

Design, implementation and testing took about one person/week spread over about two weeks calendar time.

Discussion

Octree optimization should be useful if the interesting regions of a computational space are somewhat sparse and clumpy. Isosurfaces and function mapped cutting planes fit this description. There exist pathological isosurfaces that fill nearly the entire space where performance

spaces particularly if the interesting regions are sparse and clumpy. Furthermore, implementation is quick and easy.

Acknowledgements

I'd like to thank Dr. Val Watson and Dr. Tom Lasinski at NASA Ames Research Center for supporting this work.

References

1. W. E. Lorenson and H.E. Cline, "Marching Cubes: a High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol 21, No 4, July 1987, pp 163-169.

2. UNIX Programmer' Reference Manual, Volume 1 Section 2

3. C. L. Jackins and S. L. Tanimoto, "Octtrees and Their Use in Representing Three-Dimensional Objects," *Computer Graphics and Image Processing*, Vol. 14, No. 3, p 249-270

4. D. Meagher, "Geometric Modelling Using Octree Encoding, ", *Computer Graphics and Image Processing*, Vol. 19, No. 2, 1982, pp. 129-147.

5. J. R. Woodwark, "Techniques of spatial segmentation in solid modelling," *Spacial Data Processing Using Tesseral Methods*, Collected Papers from Tesseral Workshops 1 and 2, pp. 325-30. Sept. 1986 Swindown and Reading, England. Publ: Nat. Environ. Res. Council, Swindon, England

6. J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Association of Computing Machines,* Vol. 18, pp 509-517.

7. A. S. Glassner, "Space Subdivision for Fast Ray Tracing", *IEEE Computer Graphics and Applications*, Vol. 4, No. 10, pp. 15-22.

8. G. D. Kerlick, "ISOLEV: A Level Surface Cutting Plane Program for CFD Data," Report RNR-89-006, NAS Applied Research Office, MS

Appendix A: ISOLEV Performance Data

IRIS GTX 120/16 meg

2-Jul-90

UNIX time() utility times (in seconds)

Data	Output	Opt?	n	Total		Ave		Min		Max	
				User	System	User	System	User	System	User	System
bluntfin/pressure	isosurface/solic	YES	21	30.6	0.91	1.46	0.04	0.09	0	4.8	0.37
bluntfin/pressure	isosurface/solic	NO	21	114.04	0.14	5.43	0	4.01	0	8.89	0.04
bluntfin/pressure	isosurface/solic	YES	20	27.56	0.28	1.38	0.01	0.1	0	4.49	0.06
bluntfin/pressure	isosurface/solic	NO	20	107.35	0.18	5.37	0	4.01	0	8.83	0.06
bluntfin/pressure	isosurface/dots	NO	20	18.16	0.06	0.91	0	0.85	0	1.01	0.02
bluntfin/pressure	isosurface/dots	YES	20	2.78	0.05	0.14	0	0.01	0	0.43	0.02
bluntfin/pressure	x cut/solid	NO	21	110.4	0.19	5.26	0	4.84	0	5.68	0.05
bluntfin/pressure	x cut/solid	NO	20	105.31	0.09	5.27	0	4.84	0	5.71	0.02
bluntfin/pressure	build tree		1	1.74	0.04						
bluntfin/pressure	min/max		1	1.14	0.01						
bluntfin/pressure	x cut/solid	YES	20	17.35	0.2	0.87	0.01	0.25	0	1.46	0.03
bluntfin/pressure	x cut/dots	YES	21	2.1	0.14	0.1	0	0.02	0	0.15	0
bluntfin/pressure	x cut/dots	NO	21	20.72	0.12	0.99	0	0.92	0	1.02	0.04
shuttle/pressure	isosurface/dots	YES	21	5.82	0.83	0.28	0.04	0	0	1.95	0.38
shuttle/pressure	isosurface/dots	NO	21	97.89	1.12	4.66	0.05	4.4	0.01	5.2	0.13
shuttle/pressure	isosurface/solic	NO	21	532.19	5.38	25.3	0.26	22.35	0.03	43.1	1.83
shuttle/pressure	isosurface/solic	YES	21	56.15	5.67	2.67	0.27	0.04	0	17.6	1.63
shuttle/pressure	x cut/ solid	YES	21	62.7	8.01	2.99	0.38	0.73	0.04	3.92	1.25
shuttle/pressure	x cut/ solid	NO	21	614.52	9.48	29.3	0.45	27.42	0.09	30.1	0.91
shuttle/pressure	x cut/ dots	YES	21	8.17	3.13	0.37	0.14	0.09	0	0.54	0.69
shuttle/pressure	x cut/ dots	NO	21	114.12	3.28	5.43	0.16	5.18	0.01	5.79	0.71
shuttle/pressure	build tree		1	12.89	0.29						
shuttle/pressure	min/max		1	7.9	0.14						